

# Ex6 :procedure



## Objective

The objective of this experiment is to:

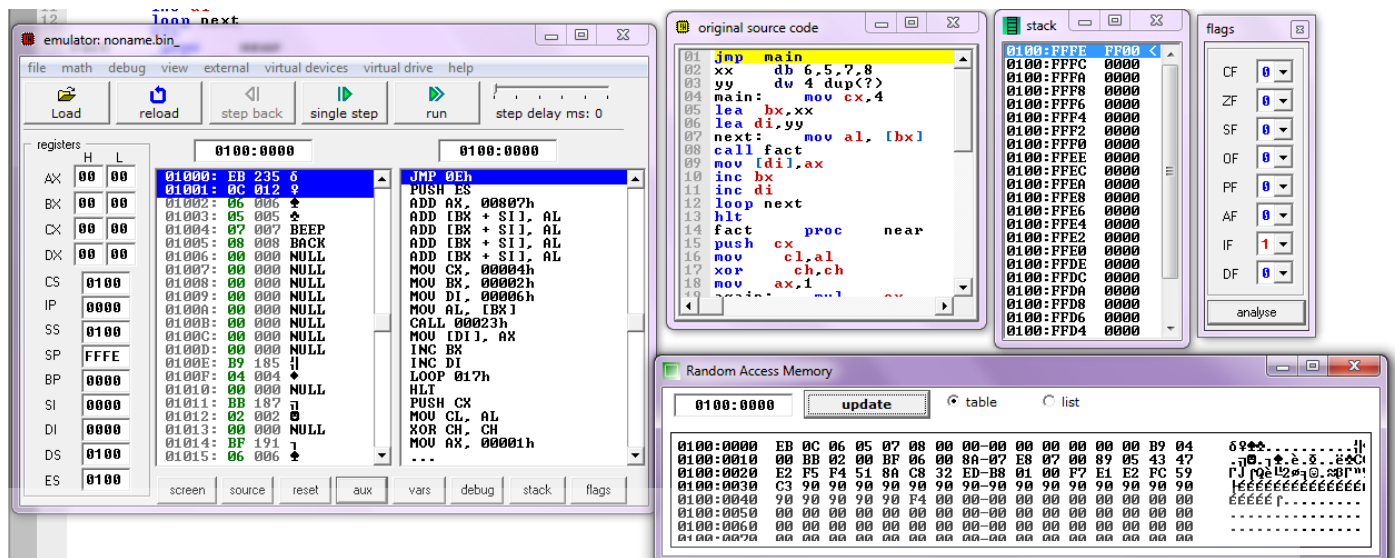
- Implement a procedure (to compute the FACTORIAL of 6, 5, 7 and 8).

## Procedure

1. Study, type, emulate and run the following program step by step (recording the contents of all affected registers, flags, memory and the stack throughout all steps):

```
        jmp main
xx      db 6,5,7,8
yy      dw 4 dup(?)
main:   mov cx,4
        lea bx,xx
        lea di,yy
next:   mov al,[bx]
        call fact
        mov [di],ax
        inc bx
        inc di
        loop next
        hlt

fact    proc near
        push cx
        mov cl,al
        xor ch,ch
        mov ax,1
again:  mul cx
        loop again
        pop cx
        ret
fact    endp
```



2. Change the value 8 in xx to 9. Emulate and run the program (press run) while watching the memory content at yy.

### Report:

1. Discuss why xx= 9 gives a wrong result in the above factorial procedure.

Because BX cannot contain more than 4 hexadecimal numbers and any number greater than 8 will need more than 4 digits in hexadecimal

2. Discuss: Why CX is pushed to the stack at the beginning of the procedure?

Because then CL and CH will change, so we pushed it first to have its value

3. Try to write and implement a procedure that computes the value of any term (up to 12 terms) in

Fibonacci Sequence and return result in AX:

$$F(n) = F(n-1) + F(n-2)$$

where  $F(0) = 0$  &  $F(1) = 1$

that is,  $F(2) = F(1) + F(0) = 0 + 1 = 1$

$F(3) = F(2) + F(1) = 1 + 1 = 2$

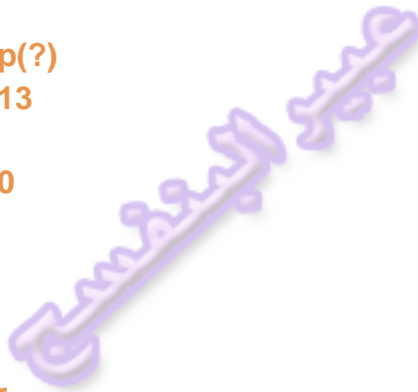
and so on

The sequence of numbers (in decimal) will look like:

$Z(n) = 0 \ 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ 13 \ 21 \ \dots$

Where  $\text{term}(0) = 0$ ,  $\text{term}(1) = 1$ ,  $\text{term}(7) = 13$  .....etc

```
jmp main
    yy    dw 3 dup(?)
main:   mov cx,13
lea di,yy
next:   mov ax,0
mov dx,0
call fact
loop next
hlt
fact    proc    near
mov [di],dx
push dx
inc di
inc ax
add dx,ax
mov [di],dx
push dx
inc di
again:  pop ax
pop dx
push ax
add dx,ax
mov [di],dx
inc di
push dx
loop again
pop cx
```



hlt  
ret  
fact  
endp

The screenshot displays the emu8086 debugger interface with the following components:

- Registers:**

Register	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00
SI	0100	
DI	0100	
BP	0000	
SP	FFFF	
IP	0000	
CS	0100	
DS	0100	
ES	0100	
- Assembly Code:**

```

01000: EB 235 6 JMP 08h
01001: 06 006  ADD [BX + SI], AL
01002: 00 000  NULL
01003: 00 000  NULL
01004: 00 000  NULL
01005: 00 000  NULL
01006: 00 000  NULL
01007: 00 000  NULL
01008: B9 185  MOV DX, 00000h
01009: 0D 013  CALL 0001Ah
0100A: 00 000  LOOP 0Eh
0100B: BF 191  HLT
0100C: 02 002  MOV [DI], DX
0100D: 00 000  PUSH DX
0100E: B8 184  INC DI
0100F: 00 000  INC AX
01010: 00 000  ADD DX, AX
01011: BA 186  MOV [DI], DX
01012: 00 000  PUSH DX
01013: 00 000  INC DI
01014: E8 232  POP AX
01015: 03 003  POP DX

```
- Stack:**

Address	Value
0100:FFFF	FF00
0100:FFFC	0000
0100:FFFA	0000
0100:FFF8	0000
0100:FFF6	0000
0100:FFF4	0000
0100:FFF2	0000
0100:FFF0	0000
0100:FFEE	0000
0100:FFEC	0000
0100:FFE8	0000
0100:FFE6	0000
0100:FFE4	0000
0100:FFE2	0000
0100:FFE0	0000
0100:FFDE	0000
0100:FFDC	0000
0100:FFDA	0000
0100:FFD8	0000
0100:FFD6	0000
0100:FFD4	0000
- Flags:**

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0
- Random Access Memory:**

Address	Value
0100:0000	EB 06 00 00 00 00 00 00-B9 0D 00 BF 02 00 B8 00
0100:0010	00 BA 00 00 E8 03 00 E2-F5 F4 89 15 52 47 40 03
0100:0020	D0 89 15 52 47 58 5A 50-03 D0 89 15 47 52 E2 F5
0100:0030	59 F4 C3 90 90 90 90 90-90 90 90 90 90 90 90
0100:0040	90 90 90 90 90 90 90 90-00 00 00 00 00 00 00
0100:0050	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0100:0060	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0100:0070	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00